Advanced Machine Learning Algorithms

Assessment 3 | Travel Airfare Data Product

Nathan Collins | Yasaman Mohammadi | Agustin Ferrari | Divgun Singh

12062131 | 24612626 | 24704114 | 24586556

Assessment 3:	**Modelling and Deriving Insights**
Type:	Group Assessment
Deliverables:	Preprocessing Jupyter Notebook (1x) Modelling Jupyter Notebooks (x4) Final Report **4000 words**
Points:	100
Due:	Sunday, 10th November, 23:59

Assessment Criteria

- *Comprehensibility, quality, reliability, robustness, and readability of code (25 points)*
- *Insightfulness and quality of results including assessment, risks, and recommended next steps from a business point of view (20 points)*
- *Appropriateness of communication style to audience in final report (summarization, quality of findings, recommendations, visualisations, readability, clarity) (20 points)*
- *Breadth of evidence of collaborative work (e.g., meeting minutes, details of contributions, etc.) (20 points)*
- *Relevance of documentation and instructions for deploying models and running applications, and robustness of data services (15 points)*

Github Repositories:

Experiments & Report

https://github.com/divgunsingh/Airline_price_modelling/tree/master

Streamlit App

https://github.com/Ferrariagustinpablo/AppStreamlit/tree/master

‘

Streamlit app:

https://appapp-9z354m9flmfnxne6d5h2tp.streamlit.app/

Section 1 Business Understanding

[1.1] Business Objective & Data Mining Goals

The objective is to build an application that provides users in the USA with quick and accurate price estimates for local flights. Users can enter details about a trip, and the app will immediately predict fares.

This product will assist in travel planning and budgeting, making flight costs clearer for consumers. This service endeavours to empower users to make more informed decisions, potentially leading to cost savings on travel expenses, providing an improved understanding of fare trends across different times and routes.

[1.2] Hypothesis

Null | *The predictive models developed for estimating local travel airfares do not significantly differ from random guessing or a simple heuristic based on average fares. In other words, the models do not add predictive power beyond what is already known or easily calculated without them.*

Alternate | *The predictive models provide significantly more accurate estimates of local travel airfares than guessing or a simple heuristic based on average fares. This implies that the models capture underlying patterns and factors affecting airfare prices and can add value by providing users with reliable fare estimates.*

Ethical Considerations

In predicting airfare travel costs, it is crucial to acknowledge that even datasets with minimal personal information can be subject to unethical use. There is a responsibility to consider how third parties might leverage the predictive outcomes. Such misuse could infringe on the privacy of individuals represented in the data or introduce bias in decision-making processes based on the model's predictions. Additionally, it is imperative to prevent unauthorised access to the data post-deployment, necessitating the implementation of stringent security measures to prevent breaches and ensure the ethical integrity of the application.

Section 2 Data Understanding and Preparation

[2.1] Understanding the Data

The dataset comprises `13521345` observations across `27` distinct variables.

Observations entail a history of airport-sorted flights across multiple companies, with features associated with each flight. These flights are pre-sorted by US-airports:

SFO: San Francisco International Airport
ATL: Hartsfield-Jackson Atlanta International Airport
BOS: Boston Logan International Airport
CLT: Charlotte Douglas International Airport
DEN: Denver International Airport
DFW: Dallas/Fort Worth International Airport
DTW: Detroit Metropolitan Wayne County Airport
EWR: Newark Liberty International Airport
IAD: Washington Dulles International Airport
JFK: John F. Kennedy International Airport
LAX: Los Angeles International Airport
LGA: LaGuardia Airport
MIA: Miami International Airport
OAK: Oakland International Airport
ORD: O'Hare International Airport
PHL: Philadelphia International Airport

Each dataset includes the target outcome "**totalFare**",

alongside other noteworthy traits, such as

- **flightDate**,
- **startingAirport** & **destinationAirport**,
- **travelDuration** & **totalTravelDistance**
- and segment data, such as **segmentsAirlineName.**

12 of the **27** features are prefixed by "**segmented**" `(Section, 2.2 Processing "segment" Columns)`. Each column represented multiple flight data, separated by ||.

*< See **appendix** for a comprehensive list of the features. >*

[2.2] Data Cleaning

Reading the complete dataframe

The data frame was formed with a script that navigated through a hierarchical file structure (list of zipped airport folders), which individually identified and extracted CSV files from each subdirectory.

The CSV files were subsequently read into a pandas DataFrame, followed by concatenation into one comprehensive DataFrame, ready for the next stages of data processing.

Checking for `missing` values

The dataset was first checked for missing values, where nulls discovered were only apparent in the `totalTravelDistance` and `segmentsEquipmentDescription` features. These columns were omitted prior to modelling `[figure 1]`.

```
Missing Values and Percentage:
                            Missing Values  Missing Percentage
totalTravelDistance                 959619            7.097774
segmentsEquipmentDescription        262676            1.942870
```

Figure 1 | *Null value counts and percentages identified within the data frame.*

It was later discovered that `segmentsDistance` possessed values such as `'None || None'`, the equivalent to further null values. This feature was likewise omitted prior to modelling.

Processing "`segment`" Columns

As 12 of the 27 features contained segmented flight data separated by "**||**", each needed to be assessed for modelling relevance and handled individually. The most applicable of these were `segmentCabinCode`, which were encoded into individual features and assigned a percentage based on their apparency in a specific row.

For example:

coach || coach || business

coach	premium coach	business	first
66.6%	0	33.3%	0

Encoding `departureDate` & `arrivalDate` Variables

Features critical to the model's input `departureDate` & `arrivalDate` were encoded into `DepartureDateDayOfWeek` and `DepartureDateMonth`.

The initial segmentsDepartureTimeRaw column contained values with distinct time entries separated by '||'. Initially, the column was divided to acquire the first datetime as it represents the departure date. After that, the day of the week and month were computed for each timestamp. An example of such a value is: '2022-05-30T15:00:00.000-04:00||2022-05-30T21:40:00.000-04:00'.

Encoding `isRefundable` to numerical

`isRefundable` was transformed into a numerical format by mapping Boolean values (True and False) to integers (1 and 0). This encoding process ensures that the feature can be effectively used as numerical input for the model.

Omitted Features

`lagId`:

> This column was omitted from the dataset as it contained identification information, its inclusion could potentially lead to overfitting.

`isBasicEconomy` and `isNonStop`:

> Including these two features is redundant as specifying the cabin type already conveys this information.

`totalTravelDistance` and `travelDuration`:

> The following features were excluded from the model because of the impracticality of relying on user input for these variables. Similarly, inferring these values presents challenges due to its complexity in calculating each segment's airport.

Supplementary features omitted due to simple input parameters requested to meet the business requirements:

```
‘totalTravelDistance,‘
‘travelDuration',
‘segmentsDepartureTimeEpochSeconds',
'segmentsArrivalTimeEpochSeconds',
'segmentsArrivalTimeRaw',
'segmentsArrivalAirportCode',
'segmentsDepartureAirportCode',
'segmentsAirlineName',
'segmentsAirlineCode',
'segmentsEquipmentDescription',
'segmentsDurationInSeconds',
'segmentsDistance'
```

Scaling

Finally, prior to modelling, all values were scaled using a `StandardScaler` to prevent weighing certain features, above others, based on their ranges.

[2.3] Feature Engineering

`Six` supplementary features were created in conjunction to the `27` provided.

These were added to assist in improving machine learning accuracy through magnifying the underlying necessity of the data. This was also performed to help mediate the exclusion of specific features, in order to meet the business requirements for a minimal input by the user.

These included;

`days_in_advance`

Examines how many days exist between the date of the flight's search and the actual flight date, (entails the `sum` of the no. days between the `flightdate` and `searchDate`) - included, as airlines often adjust pricing depending on how early or late a booking takes place.

`search_date`

Following extraction of the `day_of_the_week` , `month` , and `year` from `flightDate`, values were then converted into an integer format.

`flight_date`

Following extraction of the `day_of_the_week` , `month` , and `year` from `searchDate`, values were then converted into an integer format.

`departure_time`

Extracted from `segmentsDepartureTimeRaw`. Originally formatted as ('2022-05-20T18:58:00.000-07:00||2022-05-21T00:5...'), the 24hr time was isolated and converted to an integer format.

`segmentsCabinCode`

Extracted from `segmentedCabinCode`. This feature acts as an intermediate prior to converting cabin differentiations into percentages, (see 2.2 “Processing segment Columns”).

`numSegments`

The count of stops in each trip, derived from `segmentsCabinCode`.

`startingAirport`

One-hot encoding was applied to this feature.

`destinationAirport`

One-hot-encoding was performed on this feature as well.

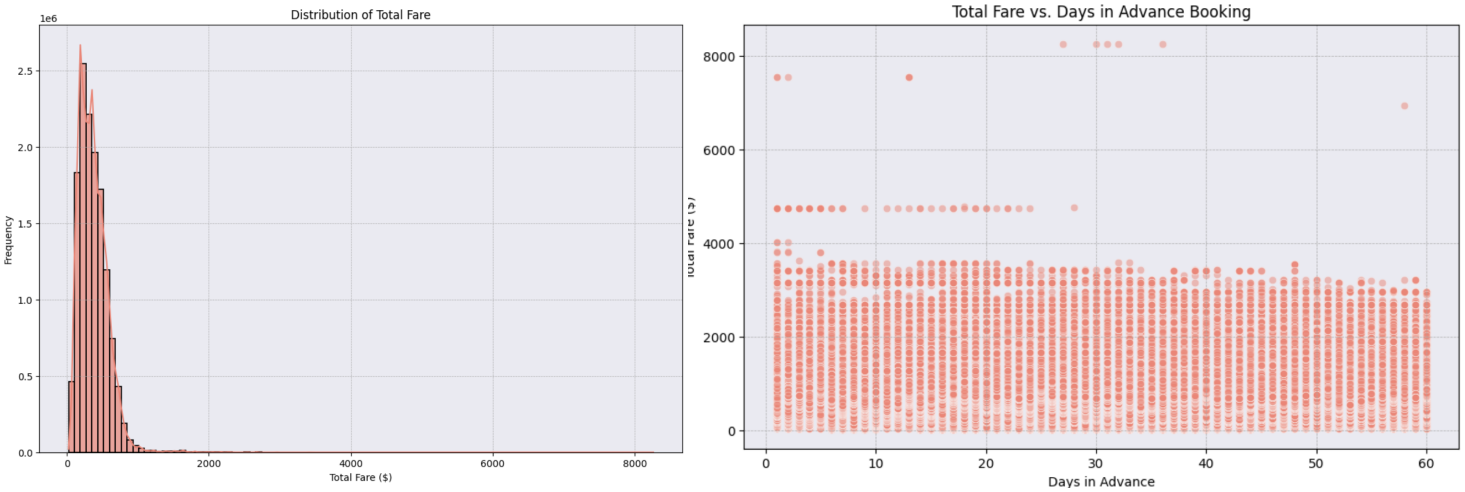
Section 3 Exploratory Analysis & Insights

count	13521345.000
mean	373.751
std	207.549
min	23.970
25%	223.470
50%	344.600
75%	487.600
max	8260.610

Understanding the target, `totalFare`

By exploring the target feature's `frequency`, coupled with `days_in_advance`, the average fare was visualised alongside its distribution.

The following revealed the average US flight within this timeframe is around $**374.00**, where fares are seen to **increase**, as the flight date approaches (figures 2 & 3).



Figures 2 & **3** | *The distribution of totalFare, followed by comparison with Days in Advance.*

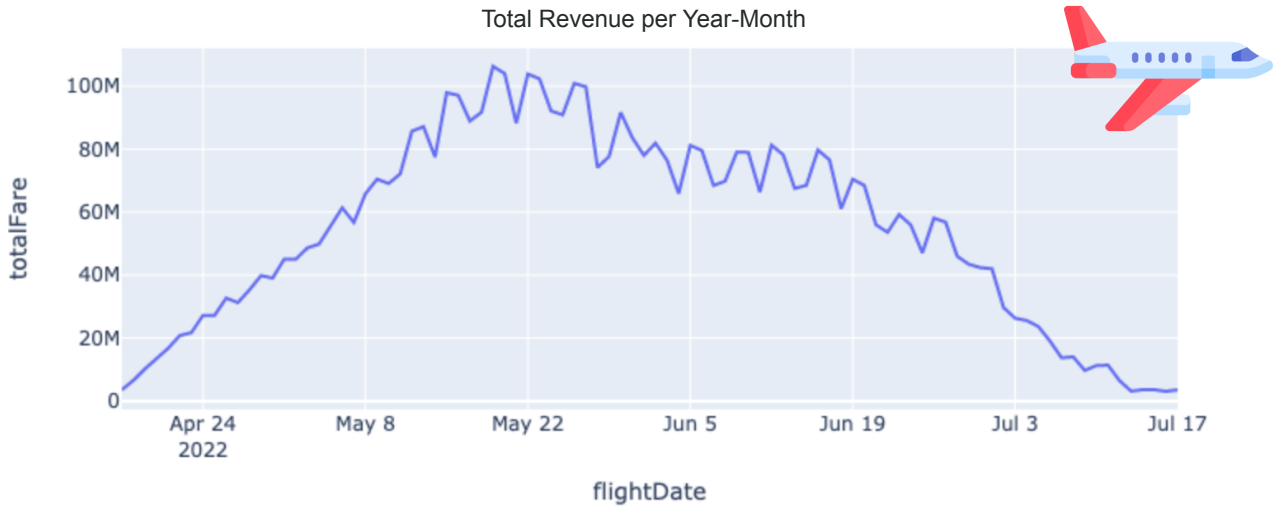
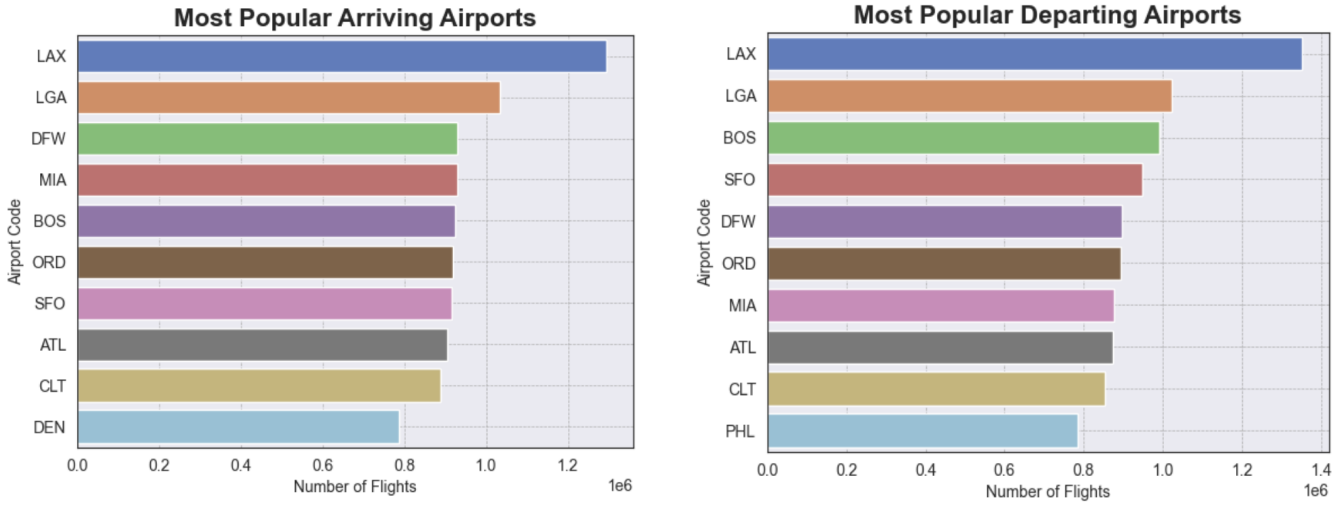


Figure 4 | *A line graph displaying the total revenue generated by all airlines.*

Understanding, `Airport`

When tallying the frequencies of `startingAirport` and `destinationAirport`, airports with the highest turnover may be interpreted. This was indicated as Los Angeles International, followed by LaGuardia (Figures 4 & 5).



Figures 5 & 6 | *Bar charts illustrating the most popular arriving and departing airports.*

These airports were additionally sorted by `frequencies` when coupled, illustrating the most popular trips among the population throughout the given time period. This was found to mirror the previous bar charts.

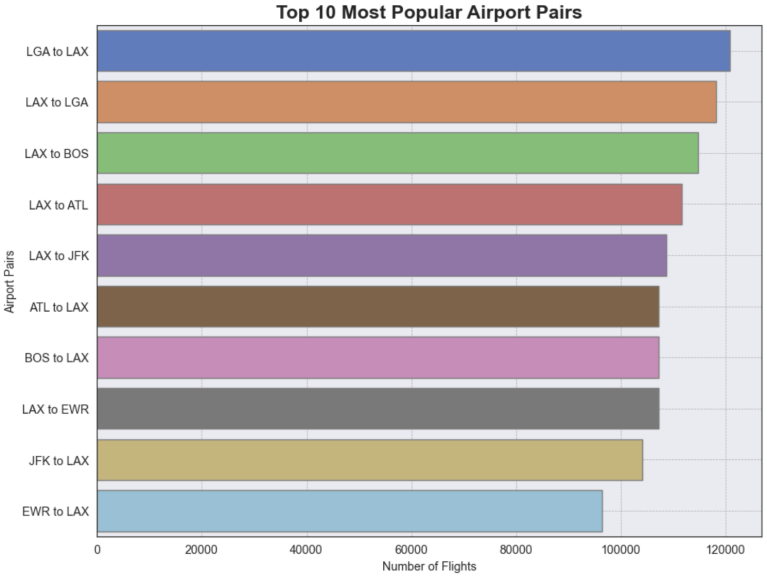
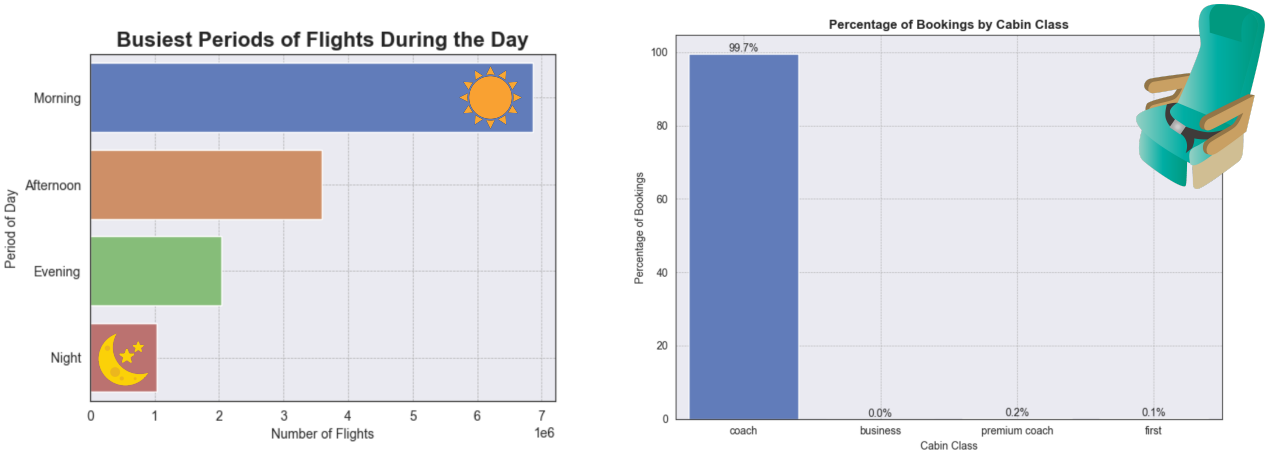


Figure 7 | *Visualising the most popular trips.*

Understanding, `popular traits`

By understanding the most popular features of each flight, insights into the cohort's routine and predictable habits may be examined. Here, almost all flights (`99.7%`) were `coach-classed` seats, with the majority taking place throughout `mornings`, (Figures 8 & 9).



Figures 8 & 9 | *Bar charts illustrating the most popular arriving and departing airports.*

The average `totalFare` by `cabin` was lastly extended to `segmentsAirlineName`, where the average cost per seat, by `cabin`, could be visualised per `airline provider`. This revealed `Hawaiian Airlines` as the most expensive `coach`, though this is likely due to Hawaii's relative distance compared to most US states.

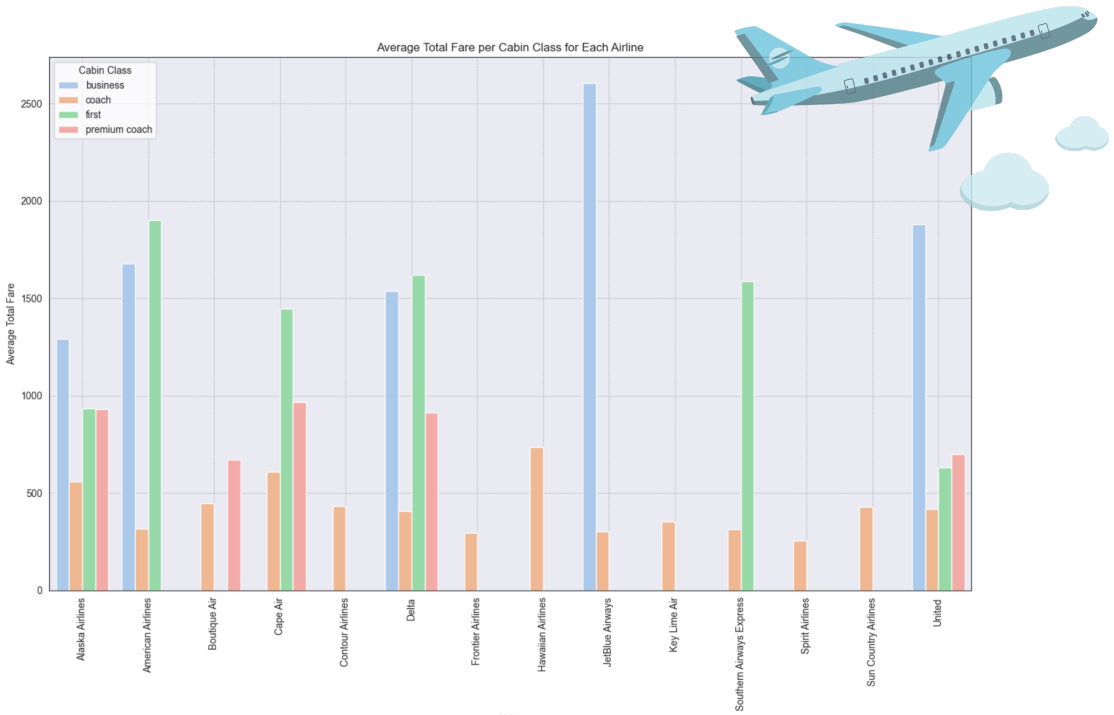
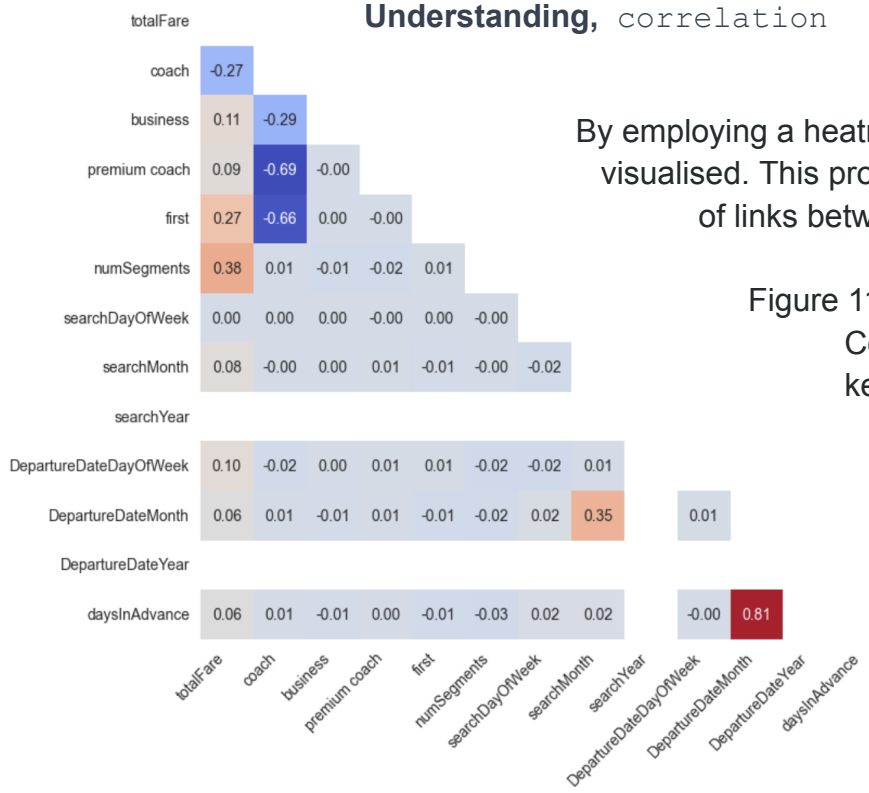


Figure 10 | *A bar chart illustrating the average totalFare, per cabin, provided by each airline.*

Understanding, `correlation`



By employing a heatmap, correlated features (r) may be visualised. This provides means for quick identification of links between criteria provided in the dataset.

Figure 11 illustrates a very limited array of Correlation between features. The key takeaway is that `coach` provides a negative correlation against `totalFare`, while `first`, indicates a complete inverse.

Figure 11 | *A correlation heatmap between all primary features of the dataset.*

Understanding, `refundability`

The last aspect of the EDA explored `refundability`'s effect on a trip's `totalFare`. Refundability was found to only be provided to `coach` tickets, where each was marked as more expensive than a `non-refundable` ticket.

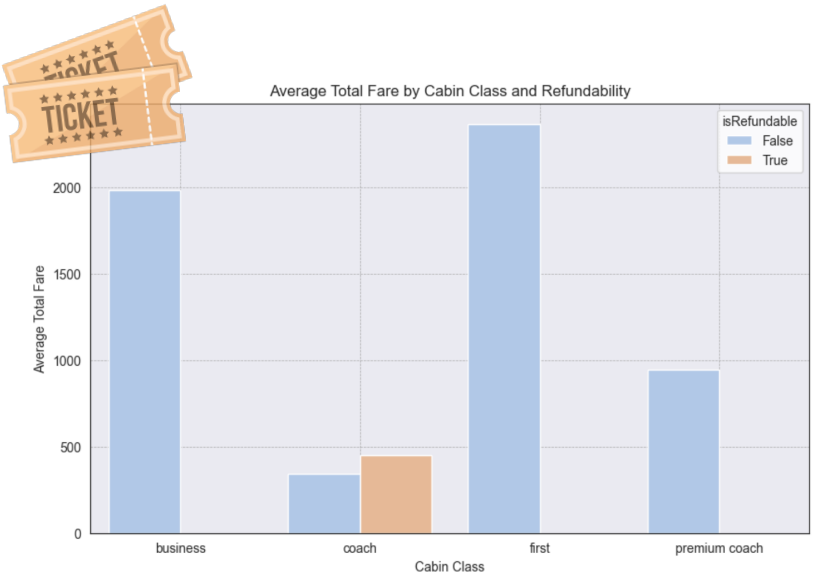


Figure 12 | *A bar chart illustrating refundable trips with their associated cabin and totalFare.*

Section 4 Modelling

Linear Model | LR

A linear model attempts to model the relationship between two or more variables with a linear equation. It operates on the assumption that a change in one variable will result in a proportional change in another variable, making it suitable for problems with linear relationships. The model provides an initial benchmark, generating a baseline for performance metrics against more complex models.

K-Nearest Neighbour | KNN

KNN is a non-parametric method used for classification and regression that makes predictions based on the proximity of data points in the feature space. KNN was utilised due to its effectiveness in capturing complex patterns without requiring any assumptions about the form of the mapping function.

Extreme Gradient Boost | XGB

XGBoost is an optimised gradient boosting library that uses decision tree ensembles to improve prediction accuracy. This model employs multiple rounds of boosting weak learners, typically decision trees, to improve the robustness and accuracy of the final model. It was picked due to its superior performance on structured data and its ability to handle various types of predictive modelling problems effectively.

Light Gradient Boosting | LGBM

LightGBM is a gradient boosting framework that uses tree-based learning algorithms and is designed for distributed and efficient training, particularly on large datasets. It was adopted for its computational efficiency and capacity to scale without accuracy loss.

Neural Network | DNN

The neural network employs a sequential architecture with 64 and 32 neurons in its input and hidden layers, respectively, using ReLU activation. Designed for regression, it minimises mean squared error using the Adam optimizer. Evaluation is based on mean absolute error, making it suitable for predicting continuous numerical outputs.

Section 5 Results

Linear Models

> Exhibited high bias with substantial mean squared error (MSE) and root mean squared error (RMSE). While the linear models outperformed the baseline they exhibited `biassed` results with a high RMSE, making them less suitable for achieving the business goal.

Table 1 | *Summary Performance Metrics of the Linear Regression Experiment.*

Model	Test MSE	Test RMSE	Train MSE	Train RMSE
Linear Regression	25246.05	158.89	25195.1	158.73
Linear Regression (Log-Transformed)	26867.85	163.91	26952.74	164.17
GLM Gamma Distribution	25932.71	161.04	25808.78	160.65

> In the Predicted vs Real Values plot for Linear Regression (figure 13), the observed trend reveals that the discrepancies between predicted and real values tend to be more substantial when the predicted values are higher. This pattern signifies an inflexible model that struggles to adapt effectively to the nuances of the dataset.

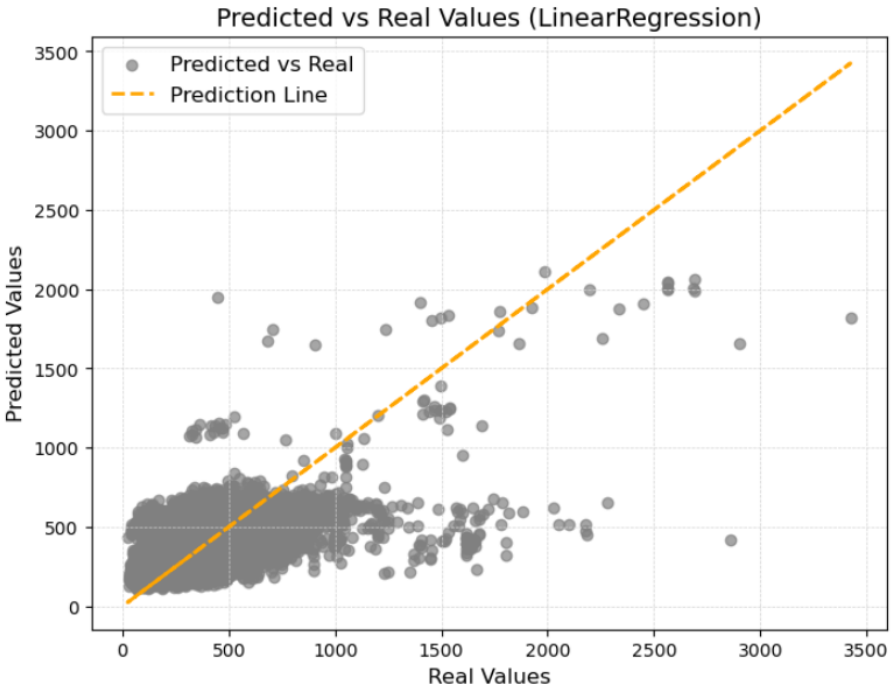


Figure 13 | *Predicted vs real values for Linear regression*

K-Nearest Neighbors (KNN)

Delivered quick performance improvements with a notable drop in loss metrics. Although KNN showed better performance compared to linear models, there were noticeable signs of potential overfitting. The best Test RMSE settled at approximately 137, determined by using hyperopt to find the most suitable number of neighbours.

Table 2 | *Summary Performance Metrics of the KNN Experiment.*

Model	Test MSE	Test RMSE	Train MSE	Train RMSE
KNN (K=5)	19692.43	140.33	13018.8	114.1
KNN Hyperopt (K=15)	18784.14	137.06	16444.8	128.24

The plot comparing predicted versus real values indicates improvement, yet there are still regions where the model struggles to comprehend the underlying function.

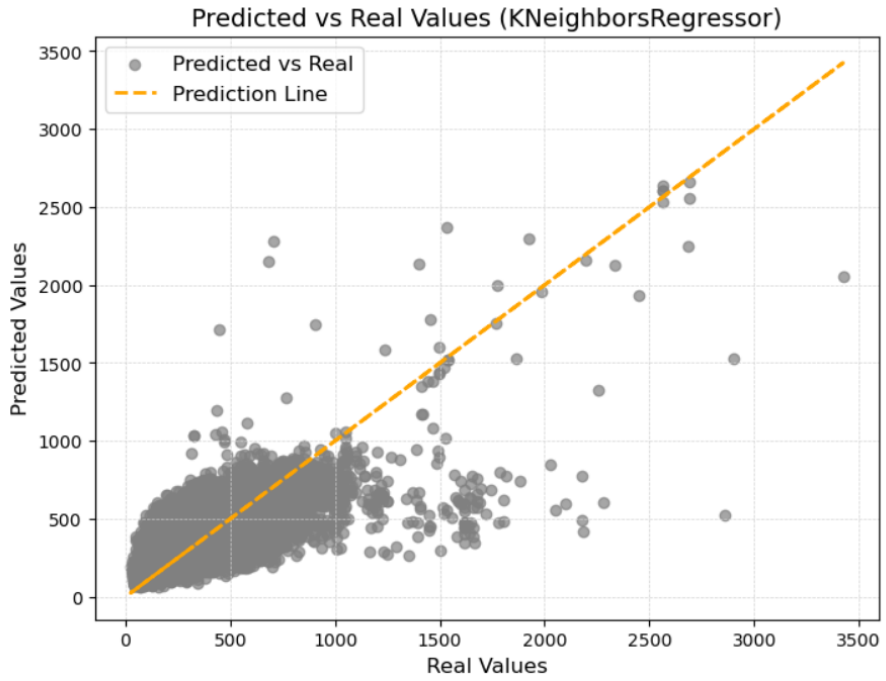


Figure 14 | *Predicted vs real values for best K-Nearest neighbours.*

XGBoost

> Although slower, it significantly enhanced performance of the last models with a considerable tendency towards overfitting. Despite attempts at hyperparameter tuning, overfitting issues could not be fully resolved, additionally the ones that were regularised displayed a lower test RMSE.

Table 3 | *Summary Performance Metrics of the XGBoost Experiment.*

Model	Test MSE	Test RMSE	Train MSE	Train RMSE
1	16534.6	128.59	15035.53	122.62
2	17106.16	130.79	16194.68	127.26
3	14425.48	120.11	7136.15	84.48
4	14079.04	118.66	4379.78	66.18
Hyperopt	14343.19	119.76	4340.16	65.88

The plot, which compares predicted values against real values, shows an overall improvement compared to previous iterations of the model. However, there are still specific regions where the model faces challenges in accurately capturing the underlying function of the data. It suggests that while there is progress, the model might not be consistently reliable across all aspects of the dataset. A considerable difference, such as 66 for the test compared to 120 for the training RMSE, indicates a substantial performance drop when the model encounters data it hasn't seen during training.

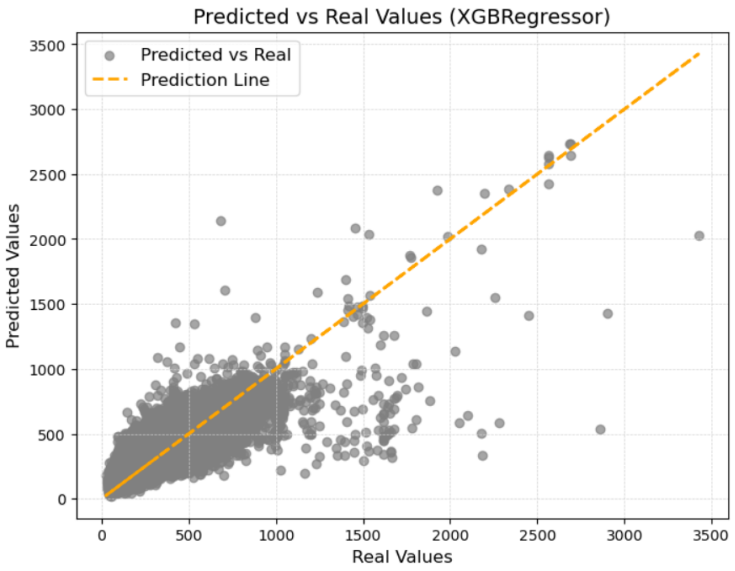


Figure 15 | *Predicted vs real values for XGBoost with best test RMSE*

Hyperparameters:

`N_estimator`

indicate how many trees are present in the forest. Generally, the larger the number of trees, the better the ability to learn the data. To find the optimal value, a grid search was used to find the optimal number of trees. However, adding too many trees can significantly slow down the training process.

`Max_depth`

In XGBOOST, the maximum depth is calculated as the longest path between the root node and the leaf node.

`Subsample`

It represents the proportion of observations to be randomly selected for each tree.

`Colsample_bytree`

Colsample_bytree determines the proportion of columns sampled when building each tree, with subsampling for each constructed tree.

LightGBM

Demonstrated exceptional speed, particularly with large datasets, allowing for the full dataset to be incorporated into the model, resulting in highly favourable outcomes.

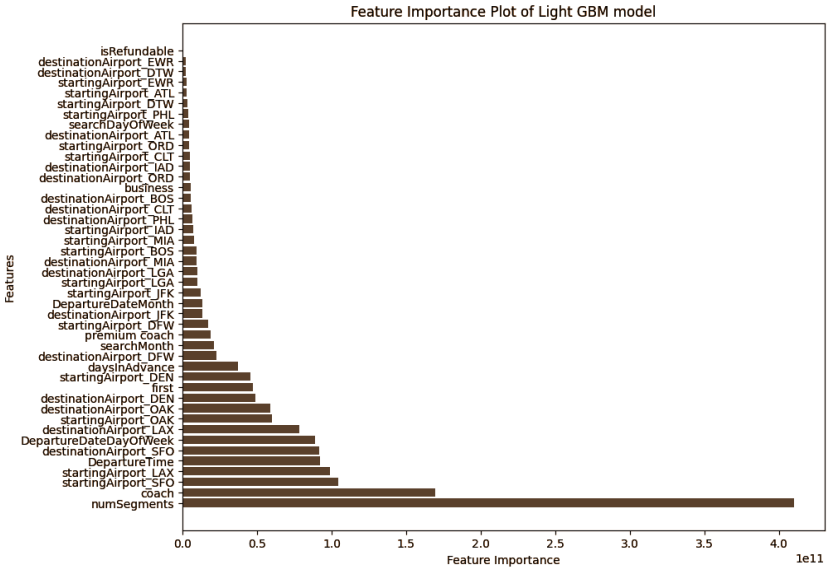


Figure 15 | *Feature Importance Plot of the Light GBM Model*

By examining the feature importance, we discovered that 'numSegments' (the count of stops in each trip) contributes a crucial role. This finding aligns with the intuition that longer trips with more stops tend to incur higher expenses.

Hyperparameters:

`N estimator`

This is the number of boosting rounds or trees trained in the LightGBM model. It is a crucial hyperparameter because it determines how many iterations the model will go through to learn from the data. Increasing the value of "n_estimators" generally allows the model to learn more complex patterns in the data, but it also makes the training process slower and may risk overfitting if set too high.

With grid search, a n estimator of 350 to 650 was selected for this experiment, and 650 was determined to be the best value.

`Max depth`

This parameter, denoting the maximum depth of the tree, is instrumental in managing model overfitting. Through grid search, a range from 5 to 100 was explored in this experiment, and the optimal value determined was 20.

Technical Performance:

The test set yielded an RMSE value of 117.33.

Deep Neural Networks (DNN) -Keras Sequential Model

Exhibited notable efficiency, especially when handling extensive datasets, enabling the integration of the entire dataset into the DNN model. Despite extended training times reported initially, the model achieved only average evaluation values, suggesting potential opportunities for optimization and improvement in performance.

Hyperparameters:

Sequential Model: This is a linear stack of layers where you can simply add one layer at a time. It's appropriate for a plain stack of layers, where each layer has exactly one input tensor and one output tensor.

Dense Layers:

First Dense Layer (64 units, ReLU activation): 64 units: The number of neurons or nodes in the layer. Higher values allow the layer to learn more complex representations, but they also increase computational requirements.
ReLU Activation: Rectified Linear Unit (ReLU) is a popular activation function that introduces non-linearity by returning the input for positive values and zero for negative values.

Second Dense Layer (32 units, ReLU activation): Similar to the first dense layer but with fewer units, potentially capturing less complex patterns.

Output Dense Layer (1 unit): Single unit because it's a regression task (predicting a continuous output). No activation function, indicating it's a linear layer, suitable for regression problems.

Technical performance

Model	Validation Set (MAE)	Validation Set (RMSE)	Test Set (MAE)	Test Set (RMSE)
DNN (Keras Sequential)	81.52799	121.19491	81.52467	121.08997

Section 6 Evaluation & Stakeholder Insights

Evaluation Metrics

RMSE(Root Mean Squared Error) is highly relevant to this project. RMSE is a popular metric for regression tasks like airfare prediction, and here is why it is chosen and how it relates to the project goals:

Root Mean Squared Error (RMSE):

RMSE measures the average squared difference between the predicted and actual values, with the result being the square root of this average.

Relevance to Project Goals:

- **Accuracy Assessment:** The primary goal of this project is to provide users with accurate estimates of travel airfare. RMSE directly addresses this goal by quantifying the accuracy of the predictions. A lower RMSE indicates that the model's predictions are closer to the actual airfares, which is crucial for providing value to users.
- **Interpretability:** RMSE provides a transparent and interpretable measure of prediction error. It gives users an idea of how much they can trust the estimated airfare, which is essential for user confidence in the application.
- **Optimisation Objective:** RMSE can be the optimization objective during the model development phase. Models can be trained to minimise the RMSE, ensuring they are fine-tuned to produce the most accurate predictions.
- **User Expectations:** Users expect airfare predictions that are as close to the actual prices as possible. RMSE directly aligns with this user expectation by providing a quantitative measure of prediction quality.

RMSE is a relevant and widely used evaluation metric for regression problems like airfare prediction. It directly supports the project's goal of providing accurate airfare estimates and offers an interpretable measure that can guide model development, comparison, and continuous improvement.

Business Impact and Benefits

To assess the impact and benefits of the final model for the business use case of estimating local travel airfare, several key aspects needed to consider:

1. **Improved User Experience:**
   The primary benefit of the model is that it enhances the user experience. Users can quickly and easily estimate their travel airfare by providing essential trip details through the Streamlit app. This convenience can lead to increased user engagement and satisfaction.
2. **Personalization:**
   Machine learning models can provide personalised fare estimates by considering specific factors like departure date, time, cabin type, and route. This personalization is challenging to achieve manually and can help users make informed decisions based on their preferences and constraints.
3. **Reduced Time and Effort:**
   Users no longer need to search multiple sources and websites to estimate airfares manually. The Streamlit app with integrated machine learning models streamlines the process, saving users time and effort.

Data Privacy and Ethical Concerns

Data Collection for this project involves gathering user information, encompassing details like origin and destination airports, departure date and time, and cabin type. It is paramount to prioritise the privacy of this data, as it may encompass sensitive information within travel itineraries, potentially containing personal and confidential data. Safeguarding this data is of utmost importance. Additionally, ethical concerns revolve around the utilisation of collected data, necessitating clear communication with users regarding data usage, storage, and potential analysis. A fundamental ethical imperative is ensuring that prediction models do not misuse user data for unintended purposes, such as profiling, targeting, or selling to third parties.

Section 7 Deployment

Integration

Model pipelines were processed and developed separately to ensure diversity in methodology, prioritising predictive capacity, accuracy and robustness. All finalised pipelines were serialised using **joblib** for seamless integration without repeated retraining.

The team allocated **Streamlit** as the platform for deployment for its simplicity and Python-native environment. A supplementary user interface was designed and developed in conjunction to accept inputs necessary for the fare prediction; these included origin, destination, date, time, and cabin type.

Functionality

The final iteration of the app was configured to load the models individually and provide an output upon user interaction. User inputs are captured through input widgets, where the criteria is passed to a model for each fare's prediction.

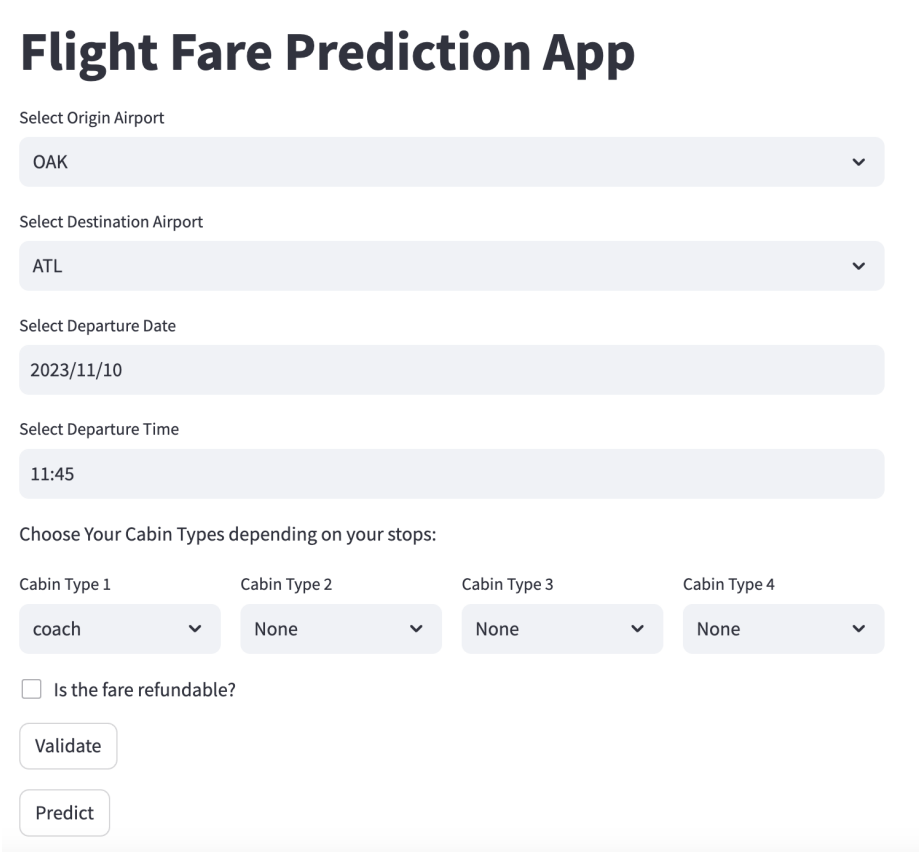
Testing

Prior to deployment, the app underwent testing to validate the level of performance of the models and gauge user experience through interaction testing. Each test involved an iterative and varied range of airports, dates, times, and cabin options to ensure reliable performance across different user inputs.

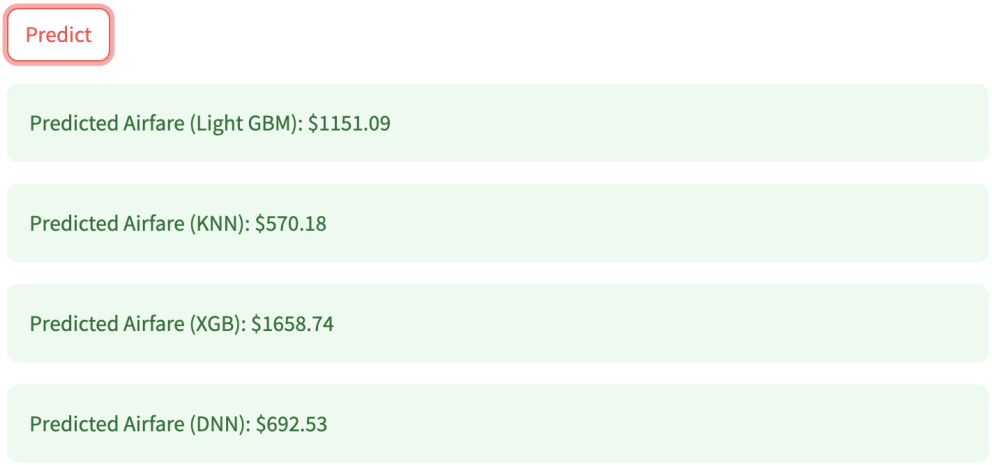
User interface

To enhance estimation accuracy, we request that users specify their preferred cabin code type (with the option of leaving it blank). This approach allows us to avoid defaulting to other features in our app, preserving accuracy. We incorporate only user-provided features and those calculable from user inputs. Additionally, given the strong correlation observed in the feature importance plot of the LightGBM model between the number of stops in a trip and airfare price, we have specifically included this option for users. Future work involves improving the user interface for increased user-friendliness.

The app interface



The results



Section 8 Conclusion

The fare-prediction Project successfully led to the development of an application capable of providing swift and precise flight fare predictions. This achievement was made possible through the exploration and implementation of various predictive models.

The project delved into Linear Regression, K-Nearest Neighbors (KNN), XGBoost, LightGBM, and a Deep Neural Network (DNN). Each model displayed distinct characteristics, with KNN, XGBoost, and LightGBM emerging as the most accurate.

A notable challenge encountered across several models was overfitting, particularly with XGBoost and KNN, highlighting the complex nature of flight fares and predicting its wide spectrum of possibility. LightGBM stood out for its efficiency and accuracy in processing large datasets. In contrast, the DNN, despite its capability to handle extensive data, exhibited only average performance, suggesting room for further optimization.

Achieving Project Goals and Meeting Stakeholder Expectations:

The project's primary goal was to create a user-friendly application that provides accurate flight fare predictions, aiding users in efficient travel planning and budgeting. This goal was successfully met with the deployment of the final model on Streamlit, an accessible platform that enhanced user interaction and experience. The project adeptly combined advanced data science methods with practical application, effectively fulfilling the stakeholders' requirements.

Recommendations and Future Directions:

Optimising Models:
Further refinement, especially of the DNN model, is crucial to enhance accuracy and reliability.

Implementing Adaptive Learning:
Incorporating algorithms that adapt and learn from new data could significantly improve the application's accuracy over time.

Integrating User Feedback:
Actively incorporating user feedback can aid in fine-tuning both the application's functionality and its user interface.

Feature Expansion:
Adding more variables, such as data on seasonal trends, special events, or airline-specific promotions, could bolster the models' predictive capabilities.

Appendix | Dictionary, Visualisations, Evidence of Collaboration

Data Dictionary

legId: *An identifier for the flight.*

searchDate: *The date (YYYY-MM-DD) on which this entry was taken from Expedia.*

flightDate: *The date (YYYY-MM-DD) of the flight.*

startingAirport: *Three-character IATA airport code for the initial location.*

destinationAirport: *Three-character IATA airport code for the arrival location.*

fareBasisCode: *The fare basis code.*

travelDuration: *The travel duration in hours and minutes.*

elapsedDays: *The number of elapsed days (usually 0).*

isBasicEconomy: *Boolean for whether the ticket is for basic economy.*

isRefundable: *Boolean for whether the ticket is refundable.*

isNonStop: *Boolean for whether the flight is non-stop.*

baseFare: *The price of the ticket (in USD).*

totalFare: *The price of the ticket (in USD) including taxes and other fees.* **Target**

seatsRemaining: *Integer for the number of seats remaining.*

totalTravelDistance: *The total travel distance in miles. This data is sometimes missing.*

segmentsDepartureTimeEpochSeconds: *String containing the departure time (Unix time) for each leg of the trip. The entries for each of the legsare separated by '||'.*

segmentsDepartureTimeRaw: *String containing the departure time (ISO 8601 format: YYYY-MM-DDThh:mm:ss.000±[hh]:00) for each leg of the trip. The entries for each of the legs are separated by '||'.*

segmentsArrivalTimeEpochSeconds: *String containing the arrival time (Unix time) for each leg of the trip. The entries for each of the legs are separated by '||'.*

segmentsArrivalTimeRaw: *String containing the arrival time (ISO 8601 format: YYYY-MM-DDThh:mm:ss.000±[hh]:00) for each leg of the trip. The entries for each of the legs are separated by '||'.*

Data Dictionary (cont.)

segmentsArrivalAirportCode: *String containing the IATA airport code for the arrival location for each leg of the trip. The entries for each of the legs are separated by '||'.*

segmentsDepartureAirportCode: *String containing the IATA airport code for the departure location for each leg of the trip. The entries for each of the legs are separated by '||'.*

segmentsAirlineName: *String containing the name of the airline that services each leg of the trip. The entries for each of the legs are separated by '||'.*

segmentsAirlineCode: *String containing the two-letter airline code that services each leg of the trip. The entries for each of the legs are separated by '||'.*

segmentsEquipmentDescription: *String containing the type of airplane used for each leg of the trip (e.g. "Airbus A321" or "Boeing 737-800"). The entries for each of the legs are separated by '||'.*

segmentsDurationInSeconds: *String containing the duration of the flight (in seconds) for each leg of the trip. The entries for each of the legs are separated by '||'.*

segmentsDistance: *String containing the distance travelled (in miles) for each leg of the trip. The entries for each of the legs are separated by '||'.*

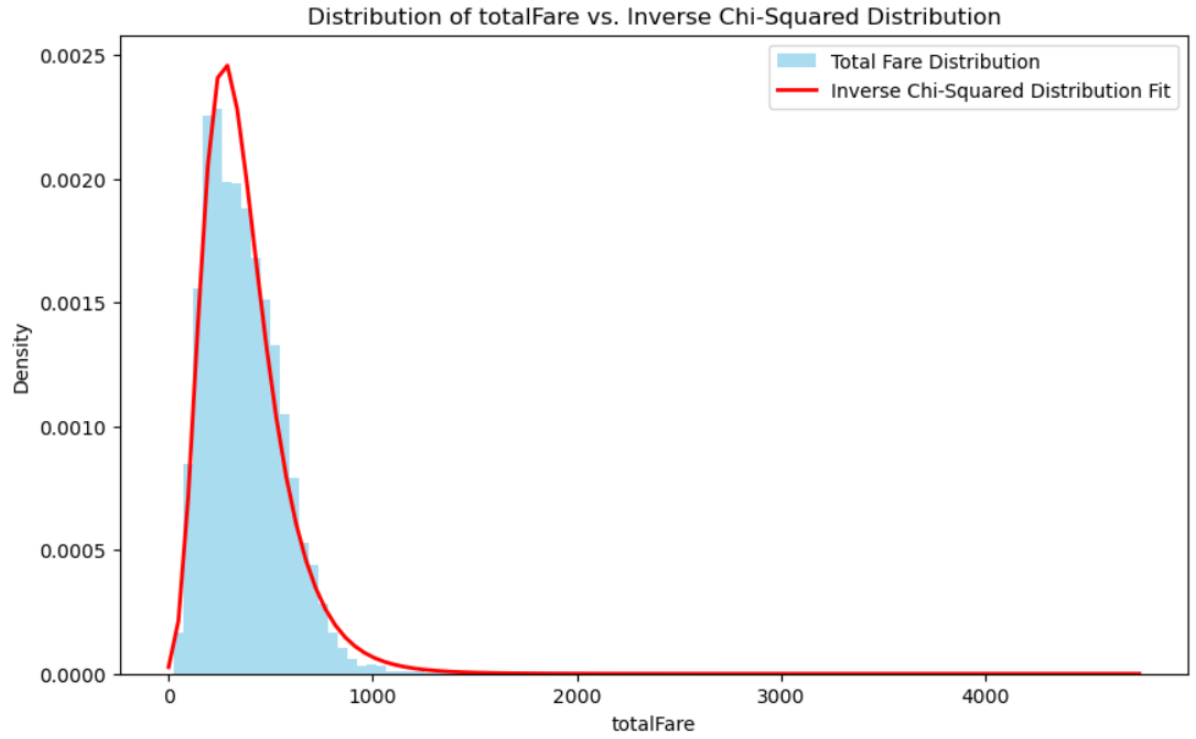
segmentsCabinCode: *String containing the cabin for each leg of the trip (e.g. "coach"). The entries for each of the legs are separated by '||'.*

Airport List

SFO: San Francisco International Airport
ATL: Hartsfield-Jackson Atlanta International Airport
BOS: Boston Logan International Airport
CLT: Charlotte Douglas International Airport
DEN: Denver International Airport
DFW: Dallas/Fort Worth International Airport
DTW: Detroit Metropolitan Wayne County Airport
EWR: Newark Liberty International Airport
IAD: Washington Dulles International Airport
JFK: John F. Kennedy International Airport
LAX: Los Angeles International Airport
LGA: LaGuardia Airport
MIA: Miami International Airport
OAK: Oakland International Airport
ORD: O'Hare International Airport
PHL: Philadelphia International Airport

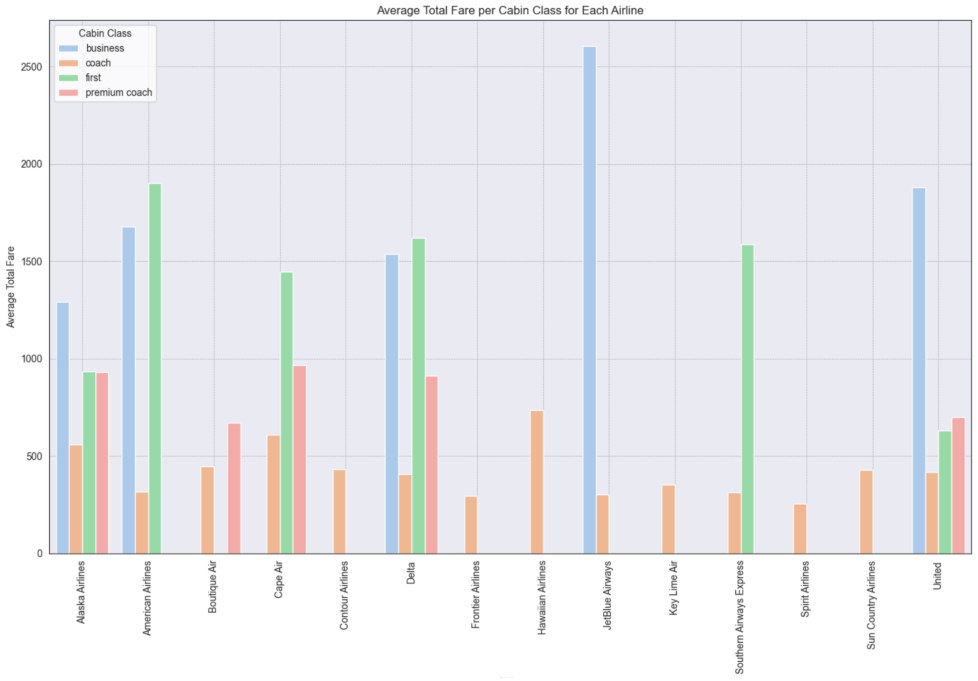
Visualisations

The presence of skewness in the distribution of totalFare is the rationale behind exploring Generalised Linear Models (GLMs).

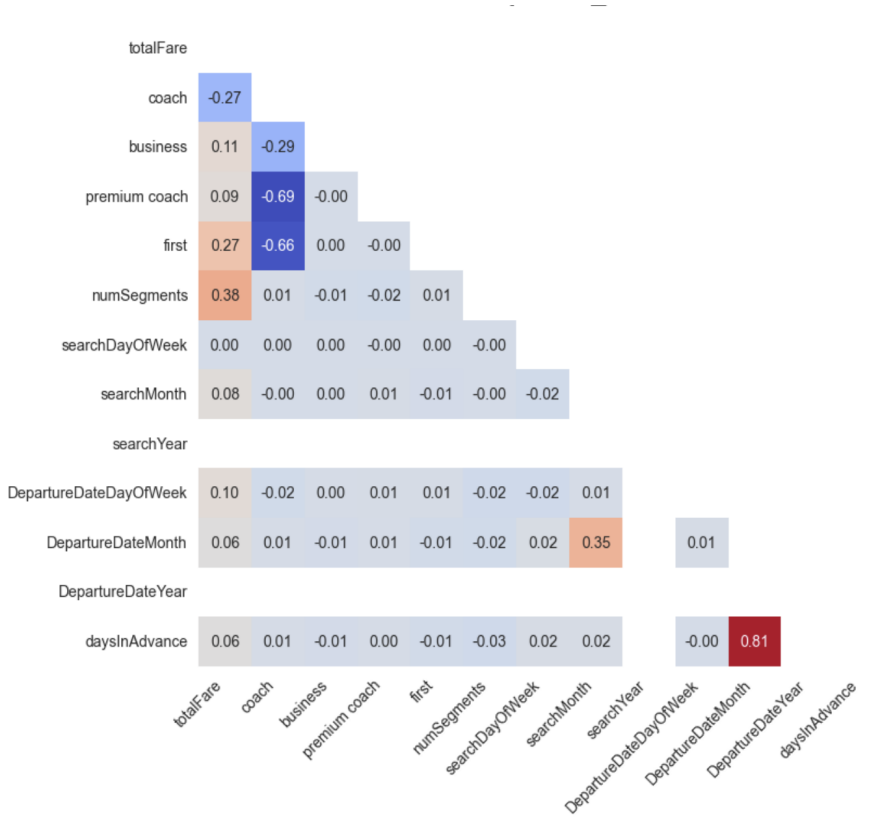


Appendix Figure 1 | *Total Fare histogram with Chi-squared fitted distribution*

Appendix Figure 2 | *Total fare by date of flight*



Appendix Figure 3 | *Average total fare per carbon class and airline*



Appendix Figure 4 | *Correlation matrix*

Evidence of Collaboration

Meeting Summaries

Location: UTS **Attendance**: 3 members

Week 1 **22**/10 Problem discussion & establishing data mining goals.
Understanding the dataset & data dictionary.
Ideation for bulk unzipping files & pre-processing.

Location: Online **Attendance**: 3 members

25/10 Finalising pre-processing phase & establishing EDA priorities.
Distribution of pre-processed dataset.
Discussed Feature Engineering, trialled new features.
Commencement of modelling, early stage progress.

Location: UTS **Attendance**: 4 members

Week 2 **28**/10 Allocation of independent modelling & evaluation tasks.
< *Group members undertook individualised approaches.* >

Offered advice and assistance where needed.
Headway on report made.

Location: UTS **Attendance**: 4 members

Week 3 **03**/11 Reviewed & discussed independent modelling approaches.
Composed descriptions of sequences for the final report.
Adjusted notebooks for variation with initial pre-processing.
Checked for errors / inconsistencies - i.e. everyone had
Standardised, applied an 80/20 split, prioritised the feature of interest, etc.

Further report development.

Location: Online **Attendance**: 3 members

05/11 Model interpretations finalised and aggregated for reporting.
Discussed deployment options and commencement.

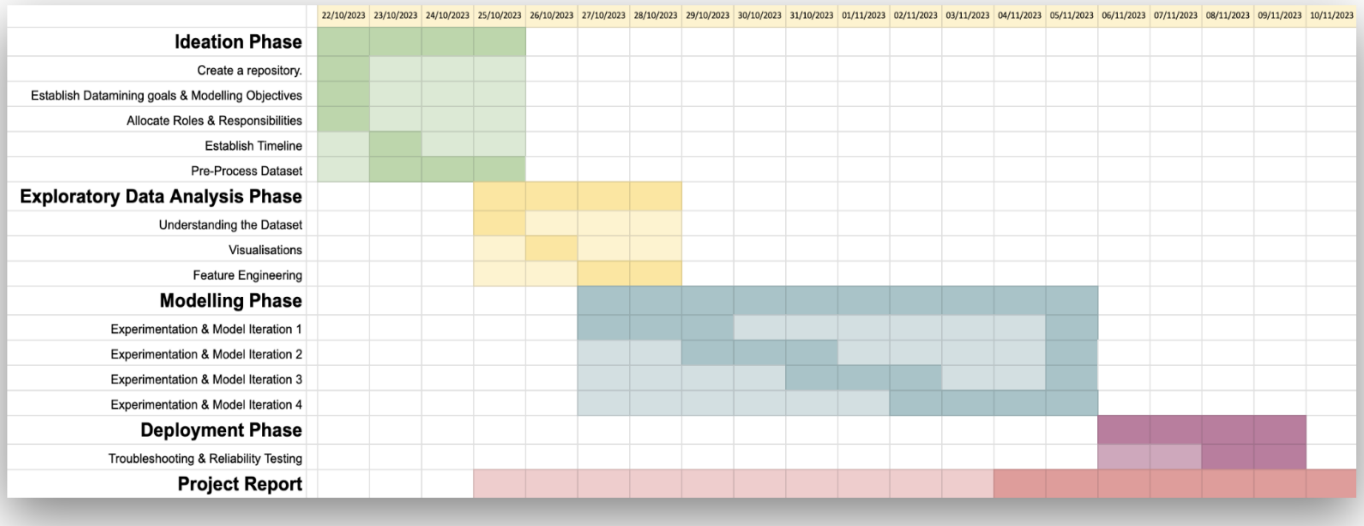
Report progress further elaborated upon.

Location: UTS **Attendance**: 4 members

Week 4 **09**/11 Deployment successful.
Prioritised troubleshooting & reliability testing
Report finalised.

10/11 Submission.

Gantt Chart | Timeline



Photos

References

Lisovyi, M. "Beware of Categorical Features in LGBM." Kaggle. Retrieved from *https://www.kaggle.com/code/mlisovyi/beware-of-categorical-features-in-lgbm.*

Prashant. "LightGBM Classifier in Python." Kaggle. Retrieved from *https://www.kaggle.com/code/prashant111/lightgbm-classifier-in-python.*

Streamlit. "API Reference: st.time_input." Streamlit Documentation. Retrieved from *https://docs.streamlit.io/library/api-reference/widgets/st.time_input.*